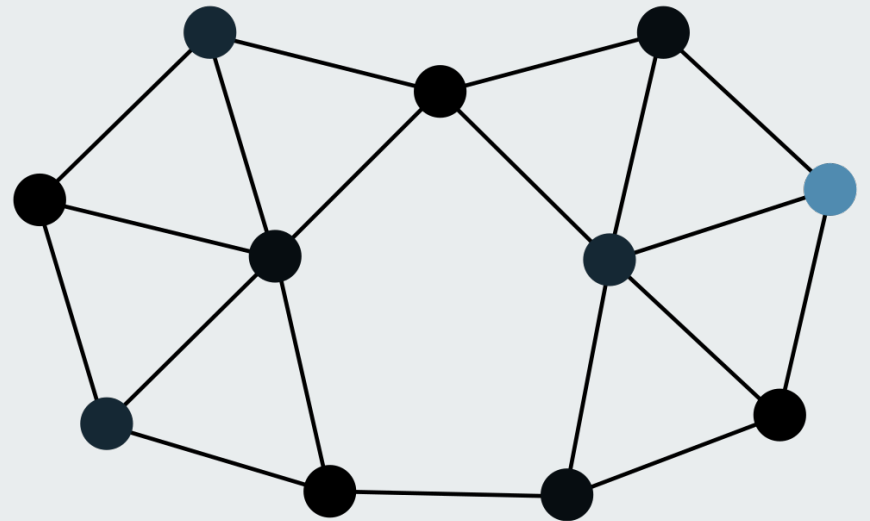


Distributed Systems

General Principles

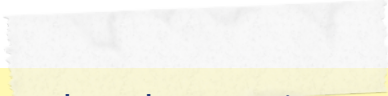




Distributed Systems

...have become commonplace in the last 20 years. Due to:

- Moore's law
- Vast improvement in networking
 - Local Area Network (LAN) **and** Wide Area Network (WAN)



Moore's Law - the observation made in 1965 that the number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented. Moore predicted that this trend would continue for the foreseeable future.



Definition

A distributed system is a collection of independent computers that appear to users as a single system.

Important **consequences**:

1. Components are autonomous
2. Users and programs see a single system
3. Computing components can be heterogeneous
4. No assumptions are made about the network topology

Middleware



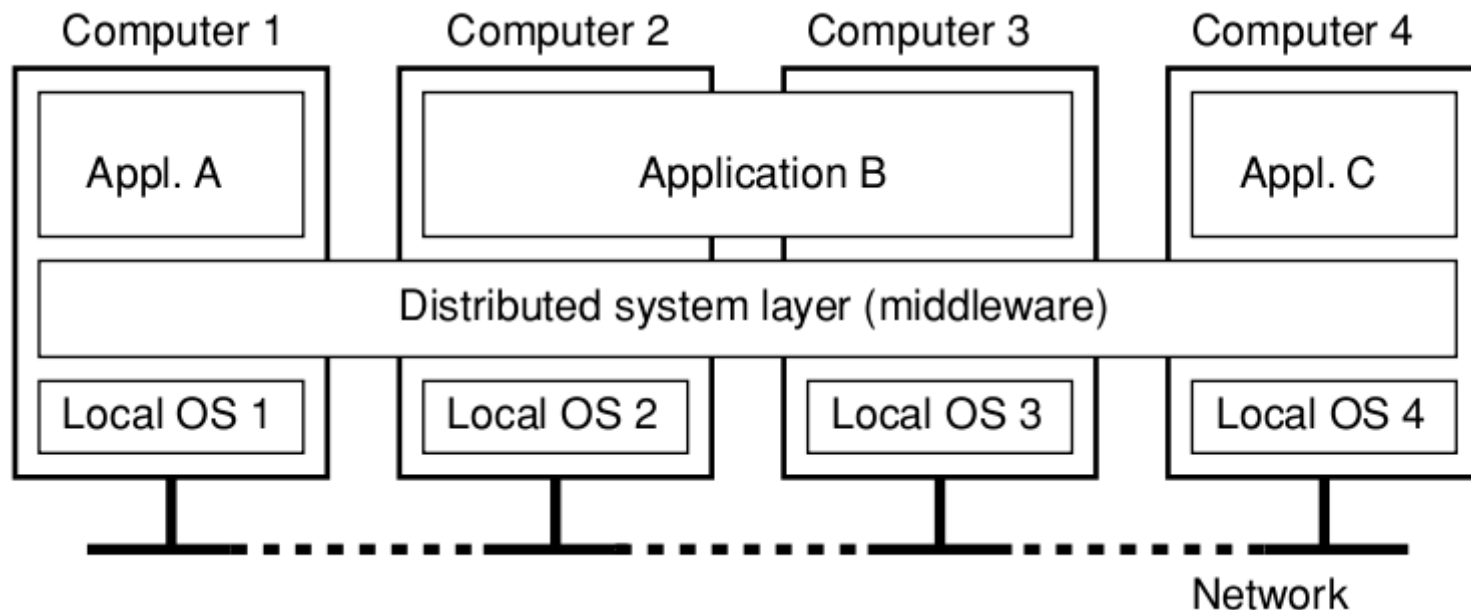
Requirements..

- Users and applications interact in a consistent and uniform way
- Easily scale
- Failures are masked/hidden from users.

Middleware

“a layer of software that is logically placed between a higher-level layer consisting of applications and users, and a layer underneath consisting of operating systems and basic communication facilities”

Hides the differences between individual nodes and OSes





Distributed System Goals

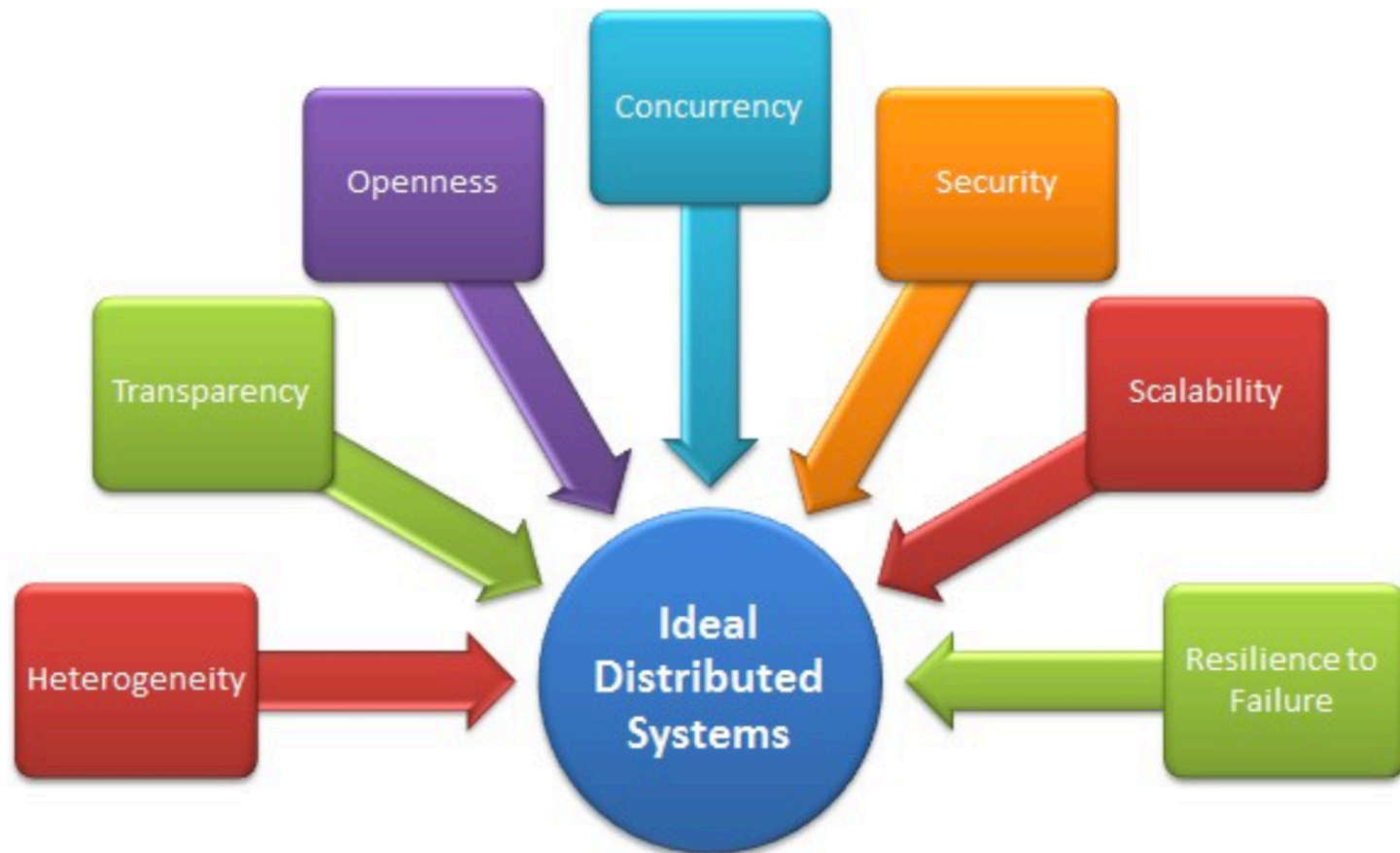
“The main goal of a distributed system is to make it easy for users and applications to access remote resources, and share them in a controlled and efficient way”

Sometimes more economical to share resources rather than replicate.

e.g. sharing printers, supercomputers, large scale storage

As sharing increases, security risks increase.

Ideal Distributed System



Distribution Transparency



Distribution Transparency

A distributed system must be able to hide the fact that processes and resources are physically distributed.

A distributed system that presents itself to users and applications as if it were a single system is said to be **transparent**.

Different transparencies:

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource



Distribution Transparency

Access transparency: Deals with differences in data representation and resource access by users.

- Should hide differences in architectures and data representation

Location transparency: users should not be able to tell where a resource is located.

- Naming services play an important role here. (e.g. DNS)



Distribution Transparency

Migration transparency: Resources can be moved without affecting how those resources can be accessed

- Should not matter if they are close by or on the other side of the world

Relocation Transparency: resources can be relocated while they are being accessed without the user or application noticing

- Much stronger condition than Migration Transparency



Distribution Transparency

Replication transparency: The system should hide the fact there are multiple copies of the same resource.

- To hide this all replicas should share the same name.

Concurrency Transparency: users should be able to share resources without each other noticing that they are using the same resource

- Will require locking mechanisms particularly around write and update operations



Distribution Transparency

Failure transparency: The user should not notice that a resource fails to work properly and the system recovers from that failure

- *“You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done”* - Leslie Lamport

Masking failures is difficult and in some cases impossible.

- Difficult to differentiate between a slow resource and a failed one.



Distribution Transparency

There will be situations where hiding all distribution is not a good idea (e.g. timezones)

Particularly when WAN networks are involved

- Considerably slower than LAN networks

Openness



Openness

“An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services.”

- Services are specified through interfaces, using an Interface Definition Language (IDL)
- Interface describes the syntax of service, not the implementation



Interface Definitions

Allows two arbitrary processes to communicate regardless of implementation

Specifications need to be:

- Complete: everything needed has been specified.
- Neutral: there is no prescription of **how** the interface should be implemented.



Openness Terms

Interoperability measures how well two different systems co-exist and work together.

Portability measures how well an application can be executed on different distributed systems without modification

Necessary for process migration to function (e.g. agent systems)

Extensibility measures how easy to add new components (without affecting the ones that are already there)



Policy and Mechanism

A **flexible** and **open** distributed system must be organised as a set of small, replaceable, adaptable components.

- Define high-level software and low-level hardware.
- Describe how they function

Otherwise it is difficult to change or replace components.



Scalability



Scalability

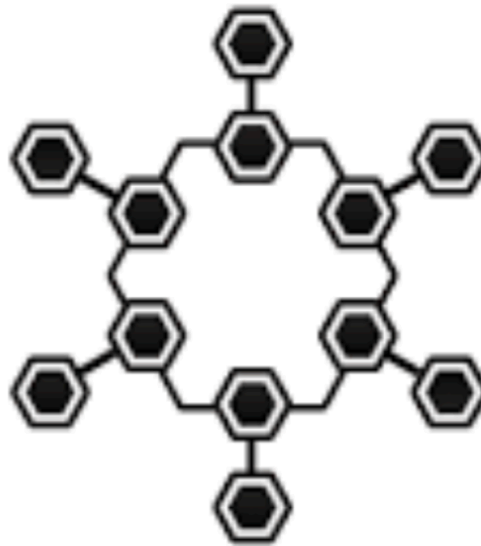
How well a system can maintain performance when resources are added.

- A system should be scalable regardless of physical distance
- A system should be administratively scalable
 - i.e. as nodes increase it is still easy to manage

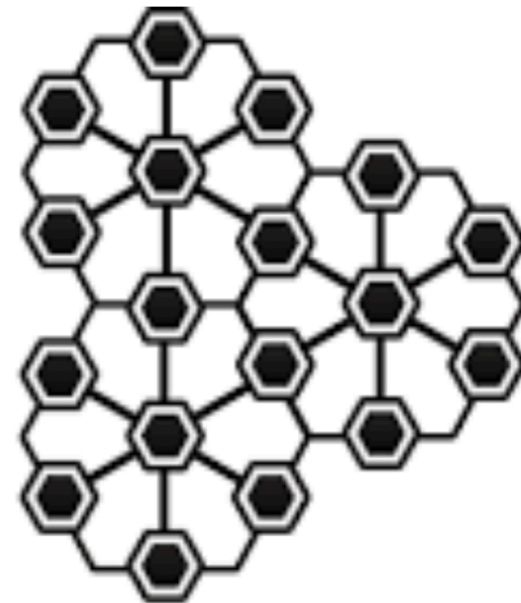
Types of Network



Centralised



Decentralised



Distributed

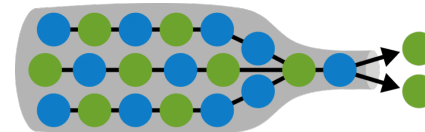
Centralised Services

Single server offers a service.

Sometimes unavoidable, hindering scalability.

For example, storage of highly confidential information

All nodes link to a single node for data, leading to network saturation (bottleneck).





Decentralised Algorithms

Have the following properties:

- No machine has complete information about system state
- Machines make decisions based only on local information
- Failure of one machine does not ruin the algorithm
- There is no implicit assumption that a global clock exists



Communication

Synchronous

- Difficult to scale
- Also known as blocking communication. Will wait until a reply is received.

Asynchronous

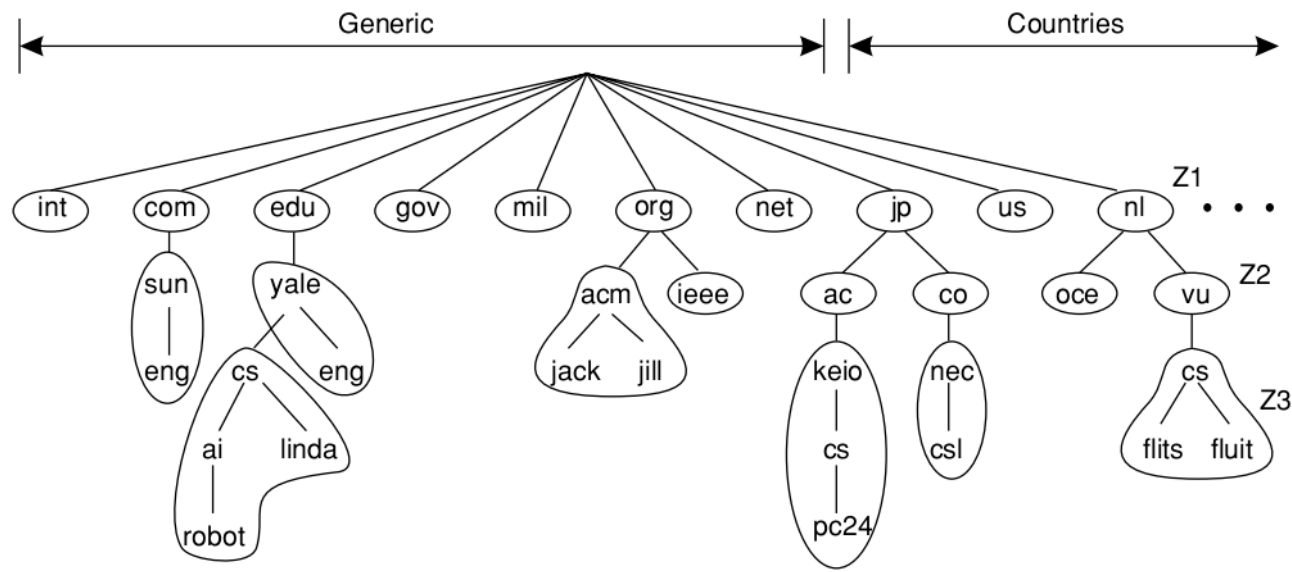
- Should be used where possible.
- Nodes can carry on working while waiting for replies.

Smaller Components

Scalability can be improved by splitting system into smaller parts, and spreading them out.

For example, the Domain Name System (DNS)

- Each step is handled by a separate server.
- Address is resolved by multiple queries





Replication

Making multiple copies of the same resource available

- Leads to load balancing and better performance

Caching is like replication, though usually held closer to the client than the original resource.

Difficulty with both: maintaining consistency



Incorrect Assumptions/Pitfalls

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is only one administrator

Categories of DS



Distributed Computing Systems

Used for high performance CPU intensive tasks.

– Types:

- Cluster computing: underlying hardware consists of similar PCs, closely connected by means of a high speed local area network. Each node runs the same operating system.
- Grid computing: federation of computer systems, each may be very different in hardware, software and network. One step further: outsourcing the infrastructure = Cloud Computing.
- Components are generally off-the-shelf commodity parts
- Networks are usually homogenous making performance predictable and scalable



Distributed Information Systems

Involve data and application that must be integrated together

- Multiple clients can access and update data
- Transactions are all or nothing. i.e. it all works, or nothing works.

For a transaction to work the following properties must be observed:

- **Atomic:** The transaction is indivisible – all or nothing
- **Consistent:** The transaction does not violate system invariants
- **Isolated:** Transactions do not interfere with each other
- **Durable:** Once a transaction commits, changes are permanent



Distributed Pervasive Systems

Such as *sensor networks* and *mobile devices*.

The network is unstable. i.e. devices move around or are added/removed

- Lacks administrative control